Does the Official Code-Review Plugin Still Help?

A First Look, Comparing It with a Simple Request on Two New AI Models

Ian J Michel

Side of My Desk LLC
research@sideofmydesk.com
https://www.linkedin.com/in/ianmichel/

Research Edition 1 • Version 1.0 • 18 November 2026

Abstract

AI coding tools can be given add-ons called plugins. A plugin is a set of written steps that tells the AI how to do a job. The steps are written for the AI models of the day. Then a newer model comes out, and few people check whether the steps still help.

This paper checks one well-known plugin: the official code-review plugin for Claude Code, made by Anthropic. We gave the AI 15 real code changes to check for mistakes. Each change was checked two ways: with the plugin, and with a simple one-paragraph request. We did this on two models, Claude Opus 5.5 and Claude Fable 5.2, twice each. That made 120 reviews. We kept a record of which model answered every request.

Without the plugin, each model found about twice as many of the known mistakes. It found 66% against 31% on Opus 5.5, and 73% against 32% on Fable 5.2. The plugin said much less. A bigger share of what it said matched a known mistake, and it cost about one and a half times as much. On none of the 15 code changes did the plugin find more of the known mistakes than the simple request.

The plugin's one advantage was smaller on the newer model. The plugin also picks its own models for its helpers. So on the newer model, the checking was still done by the older one. And the older model skipped the plugin's team process in 13 of 30 reviews. Those reviews cost a third as much and did about as well.

We also looked at the 543 things the AI reported that were not on the list of known mistakes. A second AI judged 285 of them to be real. They come to 123 separate mistakes the list does not have. When a more careful AI review looked again at 30 of those judgements, it agreed with 24.

This is a small study. The test set is public, an AI did the grading, and we did not set pass marks ahead of time. The results are a first look. They call for a bigger study. They are not a final verdict on the plugin.

1 Introduction

A plugin for an AI coding tool is mostly words. The plugin we studied has four files: a small settings file, a licence, a help page, and one page of instructions [3]. The instructions tell the AI which helpers to start, which model each helper should use, what to look for, and what to leave out. They are someone's best guess about what a model needs to be told to check code well.

That guess is made on the day the plugin is written. Models keep changing. A step that fixed a weak spot in one model may not be needed in the next. It may even get in the way. Teams add a plugin, see it work, and keep it for years. They rarely test it again.

This paper tests one idea:

> **Plugins age badly.** Steps written for an older model help less on a newer model, and may hold it back.

We had a natural way to test it. The plugin's current steps were added to Anthropic's official plugin collection on 26 October 2026. Claude Opus 5.5 came out before that, on 22 September 2026. Claude Fable 5.2 came out after it, on 9 November 2026. So Fable 5.2 is newer than the plugin. If plugins age badly, the plugin should help less on Fable 5.2 than on Opus 5.5.

What we found, in short. The biggest result was not about age. On both models, the simple request found about twice as many known mistakes as the plugin, and cost less. What the plugin did well was to say little and be right more often. The test of our idea came out half and half. One measure leaned our way clearly. The other did not. And we cannot say why, for a reason that is a finding in itself: the plugin sends its work to models it picks.

What is new here.

- A fair test of an official plugin against a simple request, on the same models, using real code changes with mistakes that people had already found. We did not find an earlier study built this way (Section 7).
- A record, for every request, of which model did the work. This includes a test showing that the tool's own summary can name the wrong model.
- A check of every reported item that was not on the known list. It suggests the list holds fewer than half of the real mistakes.
- A method that is cheap to run again each time a new model comes out. The whole study cost about $136.

This is a first look. Section 6 lists what it cannot show.

2 The plugin we tested

The plugin is the folder `plugins/code-review` in `anthropics/claude-plugins-official`, at version `9780cf4` [3]. We read all of it before running anything. It has no program code. It has one command, `/code-review:code-review`. The command tells the AI to do these steps:

1. Start a "haiku" helper to check that the code change is open, is not a draft, is not trivial, and has not been reviewed already.
2. Start a "haiku" helper to list any `CLAUDE.md` rule files.
3. Start a "sonnet" helper to sum up the change.
4. Start four checkers at once: two "sonnet" helpers to check the rule files, and two "Opus" helpers to look for bugs.
5. For each possible problem, start one more helper to double-check it. Use "Opus" for bugs and "sonnet" for broken rules.
6. Drop any problem that did not pass the double-check. Report the rest.

Haiku, Sonnet, and Opus are three sizes of Claude model, from smallest to largest. Three things about these steps matter later.

The plugin picks the models. The model you choose only runs the show. The checking and double-checking go to the models the plugin names. In our records these were `claude-haiku-5-5`, `claude-sonnet-5-5`, and `claude-opus-5-5`.

The plugin sets a high bar. The checkers are told to report only three things: code that will not run, code that "will definitely produce wrong results regardless of inputs", and broken rules they can quote. They are told not to report "potential issues that depend on specific inputs or state". General security problems are on the plugin's list of things to skip. So when the plugin leaves a problem out, it may be doing what it was told.

The help page is out of date. The help page describes a score from 0 to 100, with problems under 80 dropped. The instructions no longer use a score. The new steps arrived in one update with no version number. The next update added the number `1.0.1` and changed nothing else.

What we did not test. Anthropic has three other things with similar names: a paid Code Review service, a `/code-review` skill built into Claude Code, and a larger review that runs in the cloud. This paper is not about any of them. Our records show that all 60 plugin reviews used the plugin's command, and none of the 60 reviews without the plugin used any built-in skill.

3 How we did it

3.1 The plan

We reviewed each of 15 code changes in four ways, twice each:

What we changed	Choices
Model	Claude Opus 5.5 (`claude-opus-5-5`); Claude Fable 5.2 (`claude-fable-5-2`)
Setup	With the plugin; without it
Code change	15 real ones
Repeat	2
Total	120 reviews

A code change sent in for review is called a *pull request*. We ran the first round of all 60 reviews before starting the second round. The order inside each round was shuffled.

With the plugin. We loaded the plugin from a saved copy of version `9780cf4`. We checked that copy, file by file, against the one we had read. Then we typed `/code-review:code-review 1`. We used the plugin as it ships. Its helpers ran on the models it names.

Without the plugin. We used the same model with no plugin loaded and typed this:

> Review pull request #1 in this repository. Use the gh CLI to read it. Report every bug or defect you find in the changed code, giving the file path and line for each. Do not post anything to GitHub.

The words of this request matter. "Report every bug" asks for a long list. The plugin is told to report only what it can double-check. That difference is part of what we are measuring (Section 6).

What we kept the same. Every review used Claude Code version 2.1.325. We set the effort level to `high` ourselves, because the two models come with different default levels [1]. Both setups were allowed the same read-only commands. Each review started fresh, with no memory of the others. None could use the web.

3.2 The test set

The code changes come from the Martian Code Review Bench [22]. It has 50 real pull requests from five open-source projects: Sentry, Grafana, Cal.com, Discourse, and Keycloak. People went through them and wrote down 173 real problems. Each problem has a short description, a level of seriousness, and a type. We call these the *known issues*. Several companies test their review tools on these same pull requests. The test set's own notes say that AI tools may have seen them during training.

Five entries had a note saying something was wrong with the data. We left those out. We shuffled the other 45 inside each project before running anything, and took the first three from each. The 15 we used have 53 known issues, from 2 to 8 each.

The original pull requests are closed, and the plugin skips closed ones. So we made a fresh, open copy of each in a private place. Each copy has two saved states: the code before the change, and the code after. We checked that each copy shows the same number of changed files and changed lines as the original. Our copies do not carry the project's history, which differs from how the test set is normally used. The plugin version we tested does not read history.

3.3 Knowing which model answered

One plugin review makes many requests, to several models. Also, Claude models can refuse a request. When that happens, Claude Code normally sends the request to a different model and keeps going with that one [2, 4]. So to say "this model did this work", we need a record for every request.

We set Claude Code to save every request and every reply to disk, and to send us a log line for each one. We matched these up. For each request we then knew which model was asked, which model answered, how much text went in and out, and which helper sent it. We turned off the switch to a different model. If a request was refused, the review stopped, and we counted it as *refused*.

We tested this record-keeping first, with a stand-in for the real service that sent back answers we had scripted. When the stand-in answered as a different model, our saved files showed it correctly. Claude Code's own summary at the end still named the model we had asked for. Anyone relying on that summary would not see the switch.

Each review ended in one state: finished, refused, switched to another model, or broken. We scored only the finished ones.

3.4 Grading

We graded the way the test set's makers do, using their own instructions. A judge model first splits a review into separate findings. Then, for every finding and every known issue, it is asked whether the two describe the same problem. A known issue is *found* if any finding matches it. A finding that matches no known issue is *unmatched*.

We use two scores.

- **Share found.** Of the known issues, what share did the review find? Researchers call this *recall*.

- **Share that matched.** Of everything the review reported, what share matched a known issue? Researchers call this *precision*. We work it out the same way the test set does.

The judge was Claude Sonnet 5.5. It saw only the review and the known issue. It was not told which model or setup wrote the review. We report scores for all the known issues, and for a "Strict" group that keeps only the bug, security, timing, data, and interface types.

3.5 Checking the unmatched findings

The test set's notes say its list may be missing things. If so, a finding can be correct and still count as unmatched. To see how often, we took all 543 unmatched findings from the finished reviews. We gave each one to a second AI, the checker, along with the code change. The checker was Claude Sonnet 5.5. It answered *real*, *not real*, or *cannot tell from the change alone*. We told it to say *real* only when the change itself showed the problem. It was not told which model or setup made the finding. We then grouped the real ones, so that the same mistake reported many times counted once.

These answers are one AI's opinion, from the code change alone. To test them, we drew 30 at random: 20 that the checker had called real and 10 it had called not real. A second review judged each one again. It was done by Claude Fable 5.2, which could read the whole project's code and was not shown the checker's answer. No person has checked either set of answers.

3.6 The numbers

For each setup we add up the results across its reviews. Two reviews of the same pull request are related, so we cannot treat all 120 as separate. To show how sure we are, we give a range next to each score. We made the ranges by re-drawing the 15 pull requests at random 10,000 times and seeing how much the score moved. This follows published advice on grouping related test items [24]. Each range is the middle 95% of those results. We did not decide ahead of time how big a difference would count as real. So we report ranges and do not claim to have proved anything.

4 What we found

4.1 Did the reviews finish?

Of 120 reviews, 119 finished and one was refused. None was switched to another model. None broke. None posted anything to the pull requests. None showed any sign of having looked up the test set's answers. The judge made no errors.

The refused review was a plugin review on Fable 5.2. One of its Opus 5.5 helpers was looking at a real security hole in the code being reviewed. A safety filter stopped it and labelled the request `cyber`. Since we had turned switching off, no other model stepped in. This was a false alarm on harmless security work. Anthropic's own guide says this can happen [4], and other studies measure how often [6, 11].

4.2 How many known issues were found

Without the plugin, each model found about twice as many of the known issues (Table 1, Figure 1). The plugin reported a fifth to a quarter as many findings. A bigger share of them matched a known issue (Figure 2, Figure 3).

Table 1: Main results, using all the known issues. The range in brackets is where we are 95% sure the true value lies. "Findings" and "Unmatched" are averages per review. Cost is Claude Code's own estimate, in US dollars per review.

Model	Setup	Reviews	Share found (%)		Share matched (%)		Findings	Unmatched	Cost
Opus 5.5	Plugin	30	31.1	[19.4, 45.5]	53.2	[35.7, 71.3]	2.1	1.0	$0.60
Opus 5.5	No plugin	30	66.0	[54.7, 78.6]	21.3	[16.9, 26.2]	11.3	8.6	$0.41
Fable 5.2	Plugin	29	31.6	[19.6, 44.8]	43.1	[28.8, 57.0]	2.5	1.4	$1.75
Fable 5.2	No plugin	30	72.6	[63.0, 83.3]	26.5	[21.0, 32.0]	10.1	7.1	$1.14

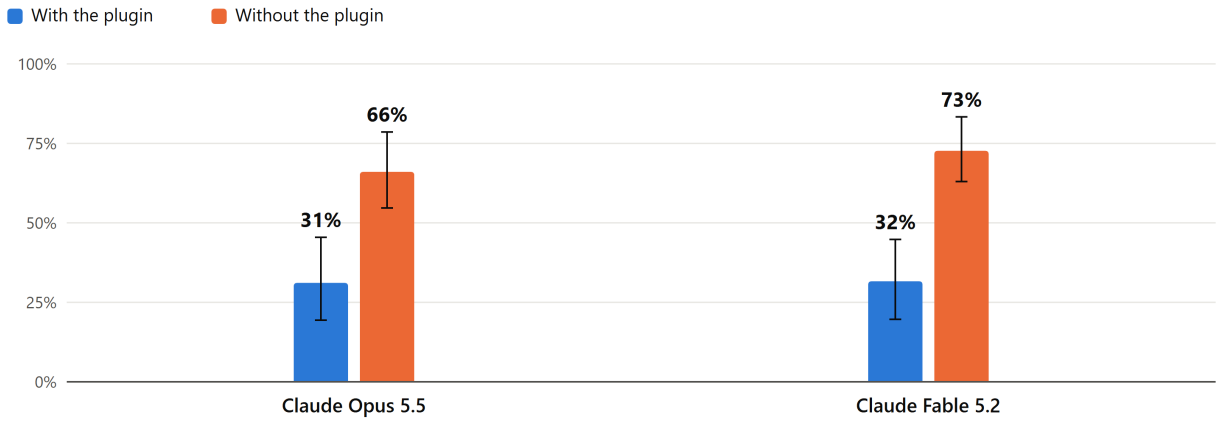


Figure 1: Share found: how much of the list of known issues each review reported. The thin lines show the 95% range.

4.3 What the plugin changed, and our idea

Table 2: Differences, in percentage points, with 95% ranges. A minus sign means the plugin did worse. The third row of each block is the test of our idea: the plugin's effect on Fable 5.2 minus its effect on Opus 5.5.

Group	Comparison	Share found		Share matched	
All	Opus 5.5: plugin − no plugin	-34.9	[-45.1, -25.0]	+31.9	[+17.9, +46.6]
All	Fable 5.2: plugin − no plugin	-41.0	[-51.0, -31.9]	+16.6	[+4.1, +27.5]
All	Difference between models (Fable − Opus)	-6.1	[-14.5, +3.7]	-15.4	[-30.1, -3.2]
Strict	Opus 5.5: plugin − no plugin	-33.3	[-44.7, -22.6]	+33.1	[+18.6, +48.3]
Strict	Fable 5.2: plugin − no plugin	-39.3	[-49.9, -30.0]	+17.6	[+6.0, +27.9]
Strict	Difference between models (Fable − Opus)	-6.0	[-12.5, +1.6]	-15.6	[-29.7, -4.5]

On both models, the plugin found far fewer known issues, and a bigger share of its findings matched. All four of those ranges stay on one side of zero, so we are confident of all four (Table 2, Figure 4).

Our idea says the plugin should help less on the newer model. The result is half and half (Figure 5):

- **Share that matched.** The plugin's lead was 31.9 points on Opus 5.5 and 16.6 points on Fable 5.2. The lead shrank by 15.4 points. The range for that drop does not include zero, so

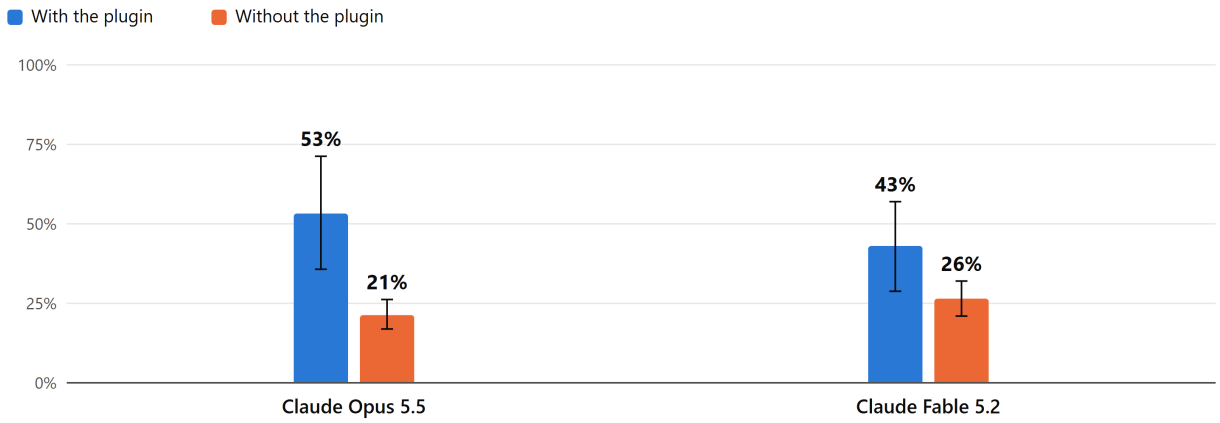


Figure 2: Share that matched: how much of what a review reported was on the list of known issues. The true figure is higher (Section 4.7).

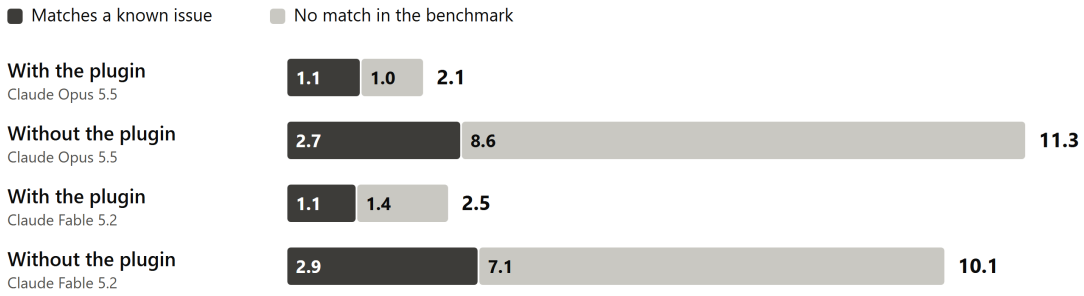


Figure 3: Average findings per review. The dark part matched a known issue. The light part did not.

we are fairly sure it is real.

- **Share found.** The plugin was behind by 34.9 points on Opus 5.5 and by 41.0 points on Fable 5.2. It fell behind by 6.1 points more. The range for that includes zero, so it could be chance.

Both numbers lean the way our idea says. The Strict group shows the same thing. But only one of the two is clear with a study this small. And it is the one most affected by the gaps in the list of known issues.

4.4 One pull request at a time

The gap is wide and steady. On none of the 15 pull requests, on either model, did the plugin find a bigger share of known issues than the simple request. They tied on two pull requests on each model (Figure 6). The gap shows up at every level of seriousness (Table 3, Figure 7).

The plugin was also less steady from one round to the next. Take the known issues a setup

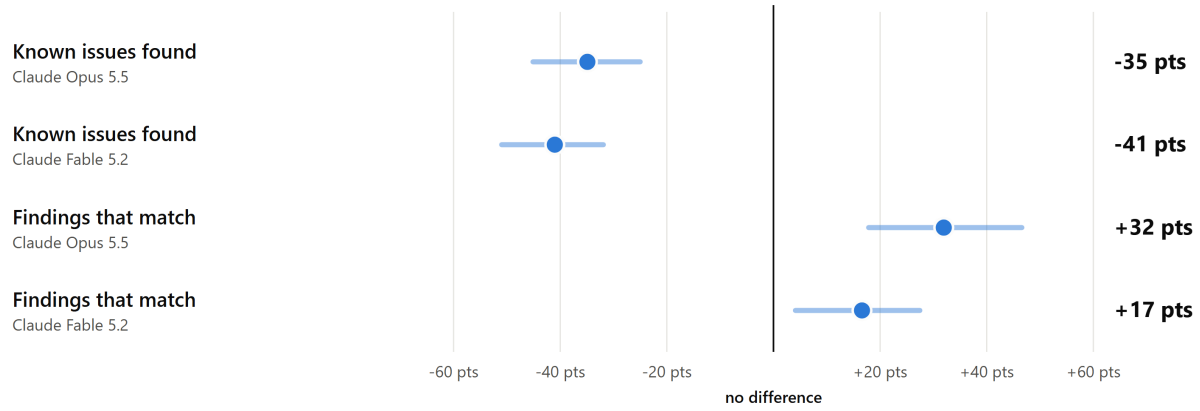


Figure 4: What adding the plugin changed on each model. Left of the line, the plugin did worse. Right of it, better.

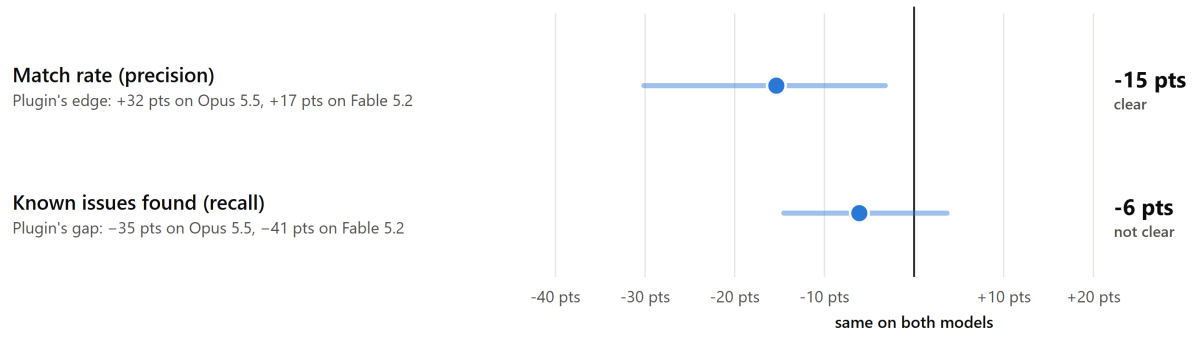


Figure 5: The test of our idea. Each dot is the plugin's effect on Fable 5.2 minus its effect on Opus 5.5. A dot left of the line supports "plugins age badly".

found at least once in its two rounds. The plugin found 63% to 65% of them both times. The simple request found 83% to 84% both times (Table 4, Figure 8). Other work has also found that AI agents give different results on repeat runs [7].

4.5 What it cost, and which models did the work

The plugin cost about one and a half times as much per review on both models (Table 5, Figure 9).

When we chose Fable 5.2, 43% of the plugin's bill was for Opus 5.5 helpers. The bug hunting and the double-checking are jobs the plugin gives to "Opus". They ran on `claude-opus-5-5`, not on the model we chose. Picking the newer model changed who ran the show. It did not change who did the checking.

Most of the cost comes from start-up. Every review and every helper begins by loading Claude Code's own long set of instructions. So the number of helpers matters more to the bill than the size of the code change.

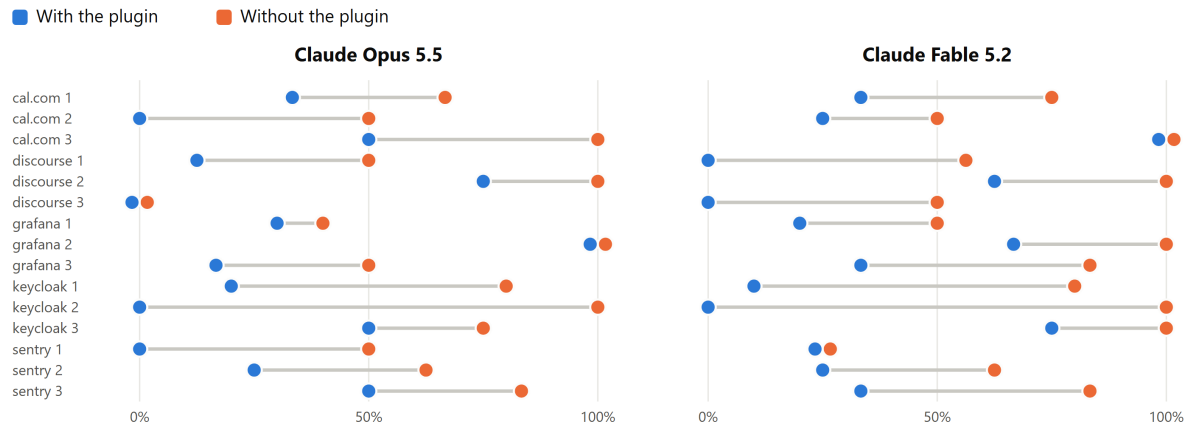


Figure 6: Share of known issues found on each pull request, with both rounds added together.

Table 3: Known issues found, out of those there were to find, by how serious the test set says each is. Counts cover all finished reviews.

	Opus 5.5		Fable 5.2	
Seriousness	Plugin	No plugin	Plugin	No plugin
Critical	3/10	6/10	5/8	8/10
High	10/24	20/24	10/23	22/24
Medium	16/40	27/40	10/35	28/40
Low	4/32	17/32	6/32	19/32

4.6 Did the AI follow the plugin?

The plugin tells the AI to hand work to helpers. But that is only an instruction. The model decides whether to follow it. Fable 5.2 started helpers in all 30 of its plugin reviews. Opus 5.5 started none in 13 of its 30, and did the review by itself (Figure 10).

When Opus 5.5 skipped the helpers, a review cost $0.28, against $0.85 with them. It found about the same share of known issues, and a bigger share of its findings matched (Table 6). We did not choose which reviews skipped the helpers. The model did. So this is a hint, not a fair test. It does fit other studies, which find that a team of AI helpers often does no better than one AI once the extra cost is counted [16, 17, 33].

4.7 Mistakes that were not on the list

There were 543 unmatched findings. The checker judged 285 of them real (52%) and 136 not real (25%). For 122 (22%) it could not tell from the code change alone (Table 7, Figure 11).

How far can the checker be trusted? The second review, with the full code to read, agreed with it on 24 of the 30 answers it looked at. Of the 20 the checker had called real, the second review agreed on 16. Of the 10 the checker had called not real, it agreed on 8 and judged 2 to be real. So about four in five of the checker's answers held up. The numbers in this section are likely somewhat too high. The second review was also done by an AI, and by one of the two models we were testing.

Many of the 285 were the same mistake reported in more than one review. Counting each mistake once gives **123 separate mistakes** that the test set does not list for these 15 pull requests. The test set lists 53 (Table 8, Figure 12). Of the 123, 27 were rated serious or very serious, and

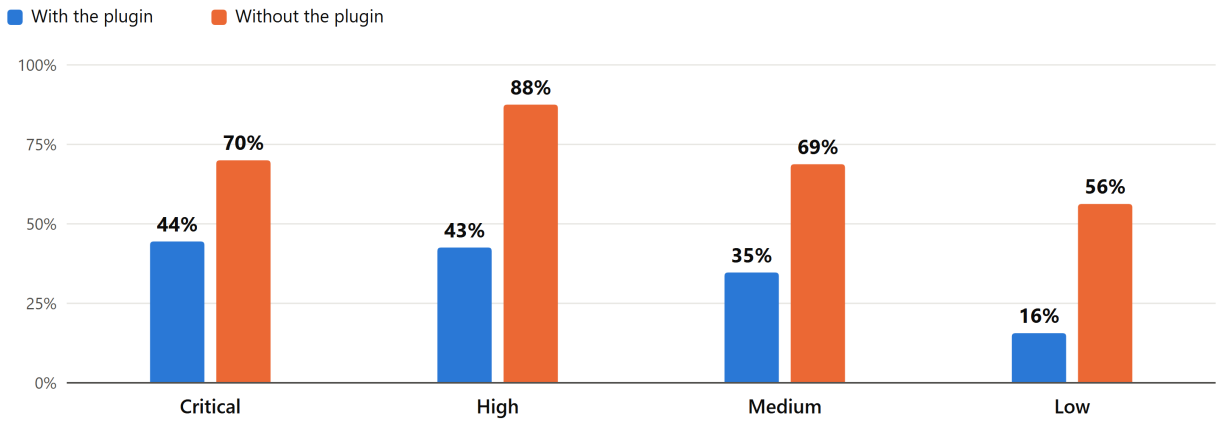


Figure 7: Share found, by seriousness, with both models added together.

Table 4: How well the two rounds agreed, counting each known issue once.

Model	Setup	Known issues	Both rounds	One round	Neither	Both, of those found
Opus 5.5	Plugin	53	13	7	33	65%
Opus 5.5	No plugin	53	32	6	15	84%
Fable 5.2	Plugin	45	12	7	26	63%
Fable 5.2	No plugin	53	35	7	11	83%

25 turned up in four or more separate reviews. Reviews without the plugin found 118 of them. Reviews with the plugin found 24.

Two things follow.

First, the "share that matched" scores in Table 1 are too low for every setup. Count a finding as good if it either matched a known issue or was judged real. Then the plugin's findings were good 81% of the time on Opus 5.5 and 72% on Fable 5.2. The simple request's were good 63% and 67% of the time. So the plugin's lead is real on Opus 5.5 and small on Fable 5.2.

Second, count good findings per review the same way. A review without the plugin gave about 7. A review with the plugin gave fewer than 2. The plugin's care did not lead to more real findings. It led to a shorter list.

Another test set, AACR-Bench, reports the same kind of gap. Its makers say that having experts go over the problems again raised the number found by 285% over what the original reviewers had written [37].

5 What it means

What the plugin does. On these pull requests the plugin worked like a filter. It gave a short list, and most of the list was right. It also missed most of what was there. The simple request gave a long list. That list had most of the known issues, more real ones that were not on the list, and more that were wrong.

Which is better depends on who reads it. A team that sees every review note as an interruption may want the filter. A team that wants mistakes caught before the code ships did better, in this study, by just asking. It also paid less.

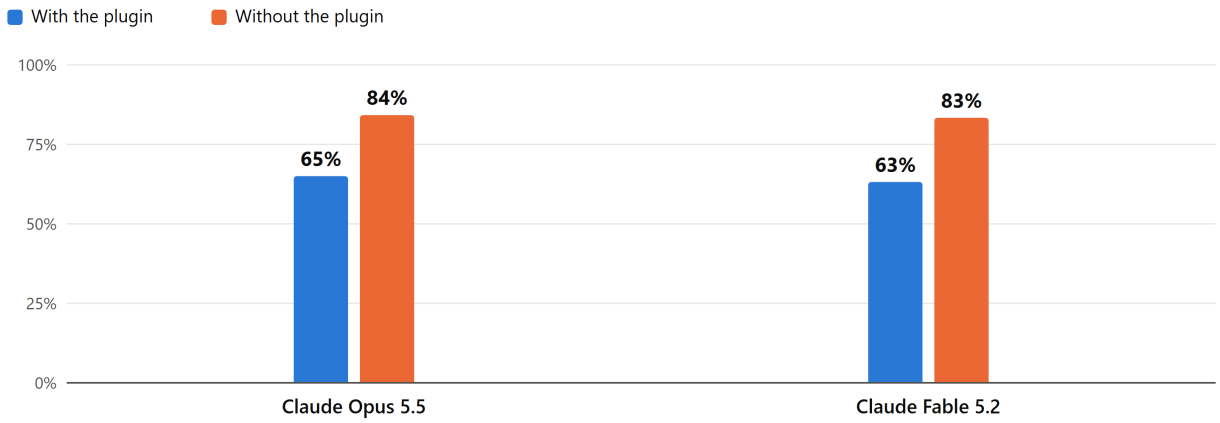


Figure 8: Of the known issues a setup found at least once, the share it found in both rounds.

Table 5: Average requests and estimated cost per review, in US dollars, split by the model that answered. Finished reviews only.

Model chosen	Setup	Requests	Fable 5.2	Opus 5.5	Sonnet 5.5	Haiku 5.5	Total
Opus 5.5	Plugin	18.4	–	0.56	0.04	<0.01	0.60
Opus 5.5	No plugin	9.8	–	0.41	–	–	0.41
Fable 5.2	Plugin	36.3	0.91	0.75	0.09	<0.01	1.75
Fable 5.2	No plugin	8.3	1.14	–	–	–	1.14

About our idea. The comparison between the two models leans toward "plugins age badly". It does not settle it. The clearer of the two results is about the share that matched. Part of the reason is that the newer model, with no plugin, seemed to score a little higher on both measures. That left the plugin less to add. We cannot be sure of that with a study this small. But it is one way a plugin can age: the model catches up with it.

A second way a plugin can age. The plugin gives bug hunting to "Opus". When we chose Fable 5.2, that work was still done by Opus 5.5. A person who moves up to a newer model does not get a newer review. Any plugin that names models for its helpers will fall behind until someone rewrites it. Our study cannot tell this apart from the steps simply fitting the newer model less well. For a user, both give the same result. For our idea, they are two different causes.

A third way. Opus 5.5 often did not follow the plugin's team process. It did the review alone, did about as well, and cost a third as much. A model that decides the steps are not needed, and skips them, is our idea seen from the model's side.

What to do with this.

- Read a plugin before you depend on it. This one picks its own models, covers less than its name suggests, and has a help page for an older design.
- Keep a record of which model answers each request, if you want to say what a model did or what it cost.

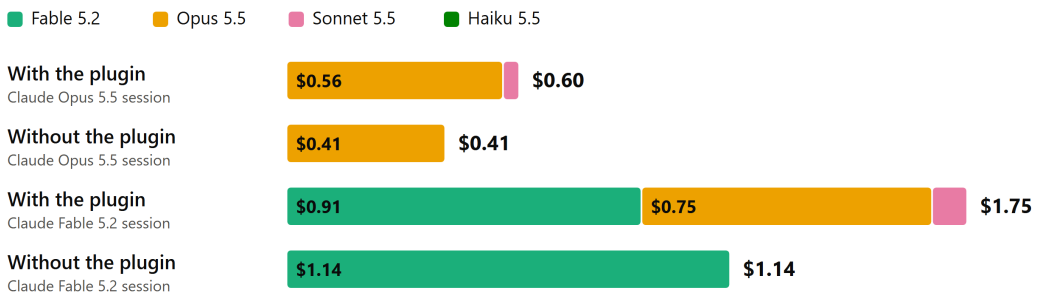


Figure 9: Estimated cost per review, split by the model that did the work.

Table 6: Plugin reviews, split by whether the AI started any helper. Scores are percentages over all known issues.

Model chosen	What it did	Reviews	Share found	Share matched	Cost per review
Opus 5.5	Launched subagents	17	32.3	46.5	$0.85
Opus 5.5	No subagents	13	29.5	68.4	$0.28
Fable 5.2	Launched subagents	29	31.6	43.1	$1.75
Fable 5.2	No subagents	0	–	–	–

- Set the effort level yourself when you compare models.
- Test your instructions again when the model changes. Here that cost about $136.

6 What this study cannot show

It is small. Fifteen pull requests give wide ranges. The main gap, in the share of known issues found, is far wider than those ranges. The differences between the two models are not, except for one.

It is a first look. We did not set pass marks ahead of time. We thought of the check on unmatched findings, and the look at skipped helpers, after the main reviews were done. Treat every result as something to test again.

The test set is public. The pull requests and the known issues were on the internet well before either model came out. AI models are known to remember public test sets [20, 27]. That could push every score up. It could also tilt things toward our idea. Without the plugin, Fable 5.2 does the review, and as the newest model it may remember more. With the plugin, Opus 5.5 does the checking either way. Test sets that are refreshed with new material avoid this [5].

The wording of the simple request. It says “report every bug”. A request for only the surest findings would likely shorten the list. It might also narrow the gap in known issues found. We tried one wording.

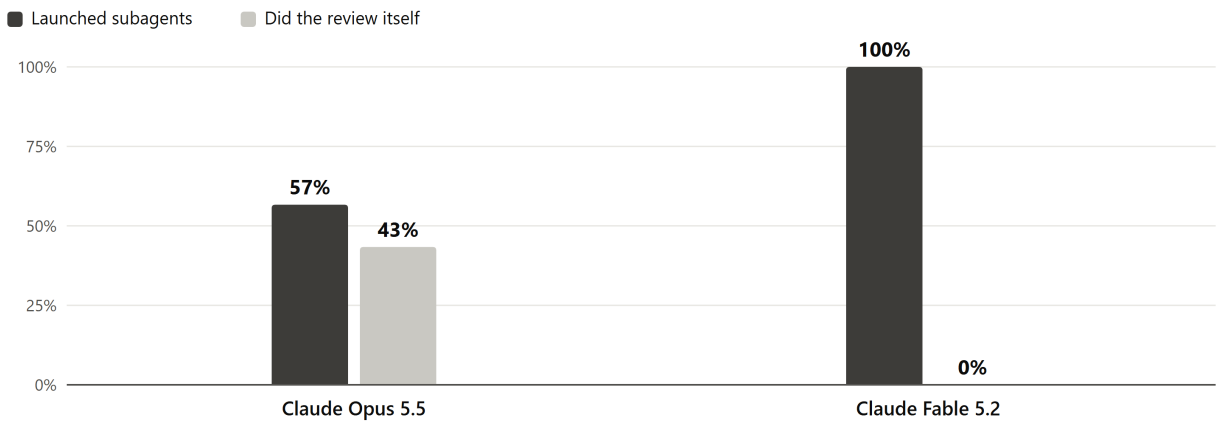


Figure 10: Share of plugin reviews in which the AI started the helpers the plugin asks for.

Table 7: The checker's answers on findings that matched no known issue.

Model	Setup	Unmatched	Real	Cannot tell	Not real
Opus 5.5	Plugin	29	17 (59%)	9 (31%)	3 (10%)
Opus 5.5	No plugin	259	132 (51%)	45 (17%)	82 (32%)
Fable 5.2	Plugin	41	21 (51%)	12 (29%)	8 (20%)
Fable 5.2	No plugin	214	115 (54%)	56 (26%)	43 (20%)

What the plugin is meant to cover. The plugin tells its checkers to skip some kinds of problem that the test set counts. So part of its low score is the plugin doing as it was told. The Strict group leaves out style, documentation, speed, and missing-test problems, and the gap is still there.

An AI did the grading. Claude Sonnet 5.5 did both the matching and the checking. AI judges are known to favour their own work and the work of related models [25, 31, 39]. Every reviewer here was a Claude model, so that should not clearly favour one setup. But the plugin does use Sonnet 5.5 for one small step. The test set's own saved results change by several points when the judge changes. A second AI review of 30 of the checker's answers agreed with 24 of them. No person checked the judge's or the checker's answers. That is the biggest thing missing.

No clean cases. Every pull request had known issues. So we did not measure how often a review raises a false alarm on code that is fine.

The two models are not a clean comparison. With the plugin, the checking was done by Opus 5.5 whichever model we chose. And Opus 5.5 skipped the helpers in 43% of its plugin reviews. So the difference between the models mixes three things: how well the steps fit, which model did the work, and whether the steps were followed.

No project history. Our copies had no history, unlike the usual use of this test set. Our numbers should not be lined up exactly against its public scoreboard.

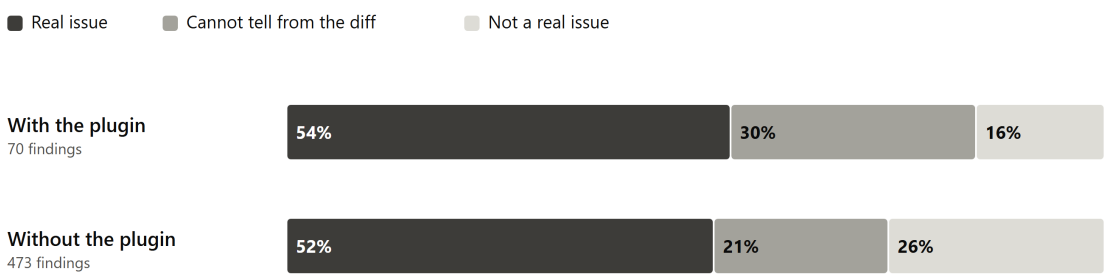


Figure 11: The checker's answers on all 543 unmatched findings, by setup.

Table 8: Separate mistakes that are not on the test set's list and that the checker judged real, by type and by which setup reported them. A mistake reported under both setups is counted in both columns.

Type	Separate mistakes	With plugin	Without plugin
bug	59	9	57
test gap	25	2	24
security	10	5	9
style	9	2	8
perf	6	1	6
data	5	2	5
doc defect	5	2	5
api	2	1	2
concurrency	2	0	2
Total	123	24	118

No one was there to give permission. The reviews ran unattended. When the AI tried a command outside the read-only set, it was turned down. A person at the keyboard would have been asked. This happened about one to three times per review, depending on the setup.

The service can change. A model's name fixes the model. It does not fix everything around it. All our reviews ran on 15 and 16 November 2026.

One plugin, one version, two models. None of this tells us about other plugins. Other studies find that well-made skills can help [18].

7 Earlier work

Do instructions keep working as models change? Others have asked this. Small changes in wording or layout can move results a lot [30]. A model sold under one name can behave differently months later [10]. Quiet updates can make existing instructions work worse [21]. Several studies find that well-known prompting tricks help less on newer models [23, 34].

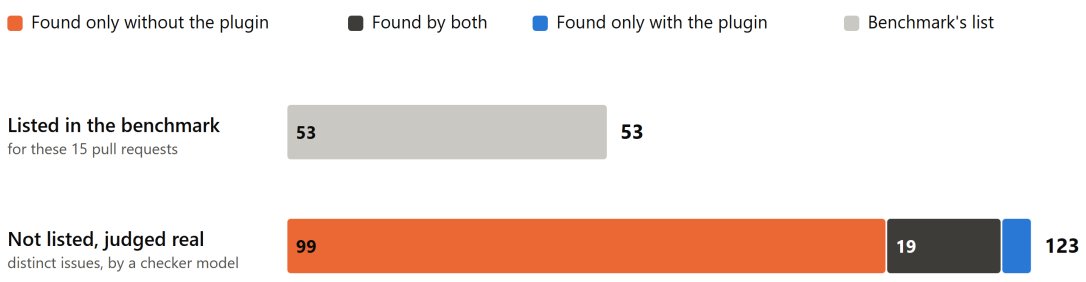


Figure 12: Mistakes on the test set's list, and separate mistakes not on it that were judged real, split by which setup found them.

The study closest to our idea is by Rudyk et al. [28]. They test five general prompting methods on pairs of older and newer models. They report that newer GPT models gain little or even lose from structured prompting, and that the pattern differs by model family. The study closest to our method is by Salsabil et al. [29]. They find that for 37% to 63% of 35 recent research tools, a single prompt on a newer model beats the tool.

Ours differs in what is tested: one fixed product, shipped by the model's maker, run unchanged on two models, against a simple request.

Do add-on skills and instruction files help? The evidence goes both ways. Gloaguen et al. [14] find that project instruction files generally do not raise success, and add over 20% to cost. Ying et al. [35] find that many popular community skills do not reliably beat an AI with no skill. Han et al. [15] test 98 software skills and find that skills give limited benefit overall. On the other side, Li et al. [18] find that carefully chosen skills raise the average pass rate from 33.9% to 50.5%. And eun Cho [12] reports that stronger models do not steadily need less structure. Our idea should be weighed against both sides.

Testing AI code review. Test sets include CodeReviewer [19], SWR-Bench [36], AACR-Bench [37], c-CRAB [38], and CR-Bench [26]. Sun et al. [32] describe a review tool used inside a large company. The trade-off we saw is known. Pereira et al. [26] report that review agents built to find everything can bury the useful findings in noise. Zeng et al. [36] find that combining several reviews raises scores a good deal. We found no study that compares a model maker's own plugin with a simple request on the same model. We found none that goes back and checks the unmatched findings.

Teams of AI helpers. The plugin is a team design with a double-check step. Cemri et al. [9] list the ways such teams fail. Kim et al. [17] report results from large gains to large losses, depending on the task. Tran and Kiela [33] find that one AI matches or beats a team when both get the same amount of thinking.

Which model is really answering? Others have studied whether a service gives you the model it names [8, 13]. We add a case inside a coding tool. The switch is one the maker documents, and the tool's own summary did not show it.

8 Conclusion

We tested Anthropic's official code-review plugin on 15 real pull requests and two current models. It found about half as many known issues as a one-paragraph request. It cost about one and a half times as much. Its list was shorter and somewhat cleaner. Its lead on that last point was smaller on the newer model. It did not use the newer model for its checking. And the older model often set its steps aside.

The idea that plugins age badly got some support. It is not proven. A stronger test would need:

- code changes made public after the models were trained;
- some code with no mistakes, to measure false alarms;
- a setup that makes every helper use one model;
- the plugin's older version;
- more models, older and newer;
- grading checked by people;
- pass marks set ahead of time.

The method is cheap enough to run each time a new model comes out. That is what the question needs.

Where to find the method and data

The testing programs, the results for every review, and the files that build this paper are public at `https://github.com/sideofmydesk/frontier-eval`. The method, the choices we made, and the limits are written up at `https://github.com/sideofmydesk/frontier-eval/wiki`. This paper and a summary are at `https://research.sideofmydesk.com`.

What we fixed in place: Claude Code version 2.1.325; plugin version `9780cf4`; test set version `e616e84`; models `claude-opus-5-5` and `claude-fable-5-2`; judge and checker `claude-sonnet-5-5`. The copies of the 15 pull requests are private, because they hold other projects' code. The public programs can make them again from the test set.

How AI was used in this work

This study was done with a great deal of help from Claude Code, Anthropic's AI coding tool. Under the author's direction, it wrote the testing programs, ran the reviews, did the analysis, searched for earlier work, and drafted this paper. The author made the design choices and is responsible for what the paper says. The tool we tested, the models that graded it, and the assistant that helped are all Claude models from one company. Readers should keep that in mind.

Funding and competing interests

The reviews cost about $136. This was paid from promotional credits that came with the author's Anthropic subscription. The author has no other financial tie to Anthropic. Anthropic had no part in this work.

Independent work by Side of My Desk LLC; views are my own, not my employer's.

Licence

This work is licensed under CC BY 4.0 (`https://creativecommons.org/licenses/by/4.0/`).

References

[1] Anthropic. Effort. `https://platform.claude.com/docs/en/build-with-claude/effort`, 2026. Claude platform documentation, accessed 15 November 2026.

[2] Anthropic. Model configuration. `https://code.claude.com/docs/en/model-config`, 2026. Claude Code documentation, accessed 15 November 2026.

[3] Anthropic. code-review plugin, claude-plugins-official. `https://github.com/anthropics/claude-plugins-official/tree/main/plugins/code-review`, 2026. Commit 9780cf4, accessed 15 November 2026.

[4] Anthropic. Refusals and fallback. `https://platform.claude.com/docs/en/build-with-claude/refusals-and-fallback`, 2026. Claude platform documentation, accessed 15 November 2026.

[5] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, et al. SWE-rebench: An Automated Pipeline for Task Collection and Decontaminated Evaluation of Software Engineering Agents. arXiv:2505.20411, 2025.

[6] Manish Bhatt, Sahana Chennabasappa, Yue Li, Cyrus Nikolaidis, Daniel Song, Shengye Wan, Faizan Ahmad, Cornelius Aschermann, et al. CyberSecEval 2: A Wide-Ranging Cybersecurity Evaluation Suite for Large Language Models. arXiv:2404.13161, 2024.

[7] Bjarni Haukur Bjarnason, André Silva, and Martin Monperrus. On Randomness in Agentic Evals. arXiv:2602.07150, 2026.

[8] Will Cai, Tianneng Shi, Xuandong Zhao, and Dawn Song. Are You Getting What You Pay For? Auditing Model Substitution in LLM APIs. arXiv:2504.04715, 2025.

[9] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, et al. Why Do Multi-Agent LLM Systems Fail? arXiv:2503.13657, 2025.

[10] Lingjiao Chen, Matei Zaharia, and James Zou. How is ChatGPT's behavior changing over time? arXiv:2307.09009, 2023.

[11] Justin Cui, Wei-Lin Chiang, Ion Stoica, and Cho-Jui Hsieh. OR-Bench: An Over-Refusal Benchmark for Large Language Models. arXiv:2405.20947, 2024. Accepted to ICML 2025, we thank everyone for their valuable suggestions and feedback!

[12] Yong eun Cho. It's Not the Capability: Harness Sensitivity Is Non-Monotone Across LLM Agent Tiers. arXiv:2605.26731, 2026.

[13] Irena Gao, Percy Liang, and Carlos Guestrin. Model Equality Testing: Which Model Is This API Serving? arXiv:2410.20247, 2024.

[14] Thibaud Gloaguen, Niels Mündler-Sasahara, Mark Niklas Müller, Veselin Raychev, and Martin Vechev. Evaluating AGENTS.md: Are Repository-Level Context Files Helpful for Coding Agents? arXiv:2602.11988, 2026.

[15] Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. SWE-Skills-Bench: Do Agent Skills Actually Help in Real-World Software Engineering? arXiv:2603.15401, 2026.

[16] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI Agents That Matter. arXiv:2407.01502, 2024.

[17] Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, et al. Towards a Science of Scaling Agent Systems. arXiv:2512.08296, 2025.

[18] Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, et al. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv:2602.12670, 2026.

[19] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, et al. Automating Code Review Activities by Large-Scale Pre-training. arXiv:2203.09095, 2022.

[20] Shanchao Liang, Spandan Garg, and Roshanak Zilouchian Moghaddam. The SWE-Bench Illusion: When State-of-the-Art LLMs Remember Instead of Reason. arXiv:2506.12286, 2025.

[21] Wanqin Ma, Chenyang Yang, and Christian Kästner. (Why) Is My Prompt Getting Worse? Rethinking Regression Testing for Evolving LLM APIs. arXiv:2311.11123, 2023.

[22] Martian. Code Review Bench. `https://github.com/withmartian/code-review-benchmark`, 2026. Commit e616e84, accessed 15 November 2026.

[23] Lennart Meincke, Ethan Mollick, Lilach Mollick, and Dan Shapiro. Prompting Science Report 2: The Decreasing Value of Chain of Thought in Prompting. arXiv:2506.07142, 2025.

[24] Evan Miller. Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations. arXiv:2411.00640, 2024.

[25] Arjun Panickssery, Samuel R. Bowman, and Shi Feng. LLM Evaluators Recognize and Favor Their Own Generations. arXiv:2404.13076, 2024.

[26] Kristen Pereira, Neelabh Sinha, Rajat Ghosh, and Debojyoti Dutta. CR-Bench: Evaluating the Real-World Utility of AI Code Review Agents. arXiv:2603.11078, 2026.

[27] Daniel Ramos, Claudia Mamede, Kush Jain, Paulo Canelas, Catarina Gamboa, and Claire Le Goues. Are Large Language Models Memorizing Bug Benchmarks? arXiv:2411.13323, 2024.

[28] Anastasiia Rudyk, Julian Oertel, and Regina Hebig. Aging of Prompt Engineering Techniques Across LLM Versions. arXiv:2608.24641, 2026. Accepted at ICSME 2026.

[29] Nahian Salsabil, Joy Saha, Simantika Bhattacharjee Dristi, Nicholas Phair, Nusrat Jahan Mozumder, Matthew B. Dwyer, and Sebastian Elbaum. What Survives the Next Model? Benchmarking LLM-Based Techniques Against Single-Prompts. arXiv:2609.00468, 2026.

[30] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting. arXiv:2310.11324, 2023.

[31] Evangelia Spiliopoulou, Riccardo Fogliato, Hanna Burnsky, Tamer Soliman, Jie Ma, Graham Horwood, and Miguel Ballesteros. Play Favorites: A Statistical Method to Measure Self-Bias in LLM-as-a-Judge. arXiv:2508.06709, 2025.

[32] Tao Sun, Jian Xu, Yuanpeng Li, Zhao Yan, Ge Zhang, Lintao Xie, Lu Geng, Zheng Wang, et al. BitsAI-CR: Automated Code Review via LLM in Practice. arXiv:2501.15134, 2025.

[33] Dat Tran and Douwe Kiela. Single-Agent LLMs Outperform Multi-Agent Systems on Multi-Hop Reasoning Under Equal Thinking Token Budgets. arXiv:2604.02460, 2026.

[34] Guoqing Wang, Zeyu Sun, Zhihao Gong, Sixiang Ye, Yizhou Chen, Yifan Zhao, Qingyuan Liang, and Dan Hao. Do Advanced Language Models Eliminate the Need for Prompt Engineering in Software Engineering? arXiv:2411.02093, 2024.

[35] Jiahao Ying, Boxian Ai, Wei Tang, Siyuan Liu, and Yixin Cao. OpenSkillEval: Automatically Auditing the Open Skill Ecosystem for LLM Agents. arXiv:2605.23657, 2026.

[36] Zhengran Zeng, Ruikai Shi, Keke Han, Yixin Li, Kaicheng Sun, Yidong Wang, Zhuohao Yu, Rui Xie, et al. SWR-Bench: Assessing LLM Performance in Real-World Code Review Comment Generation. arXiv:2509.01494, 2025.

[37] Lei Zhang, Yongda Yu, Minghui Yu, Xinxin Guo, Zhengqi Zhuang, Guoping Rong, Dong Shao, Haifeng Shen, et al. AACR-Bench: Evaluating Automatic Code Review with Holistic Repository-Level Context. arXiv:2601.19494, 2026.

[38] Yuntong Zhang, Zhiyuan Pan, Imam Nur Bani Yusuf, Haifeng Ruan, Ridwan Shariffdeen, and Abhik Roychoudhury. Code Review Agent Benchmark. arXiv:2603.23448, 2026.

[39] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685, 2023.